

프로그래밍 실습 #9

2023 년 11 월 5 주차

채진석 교수님

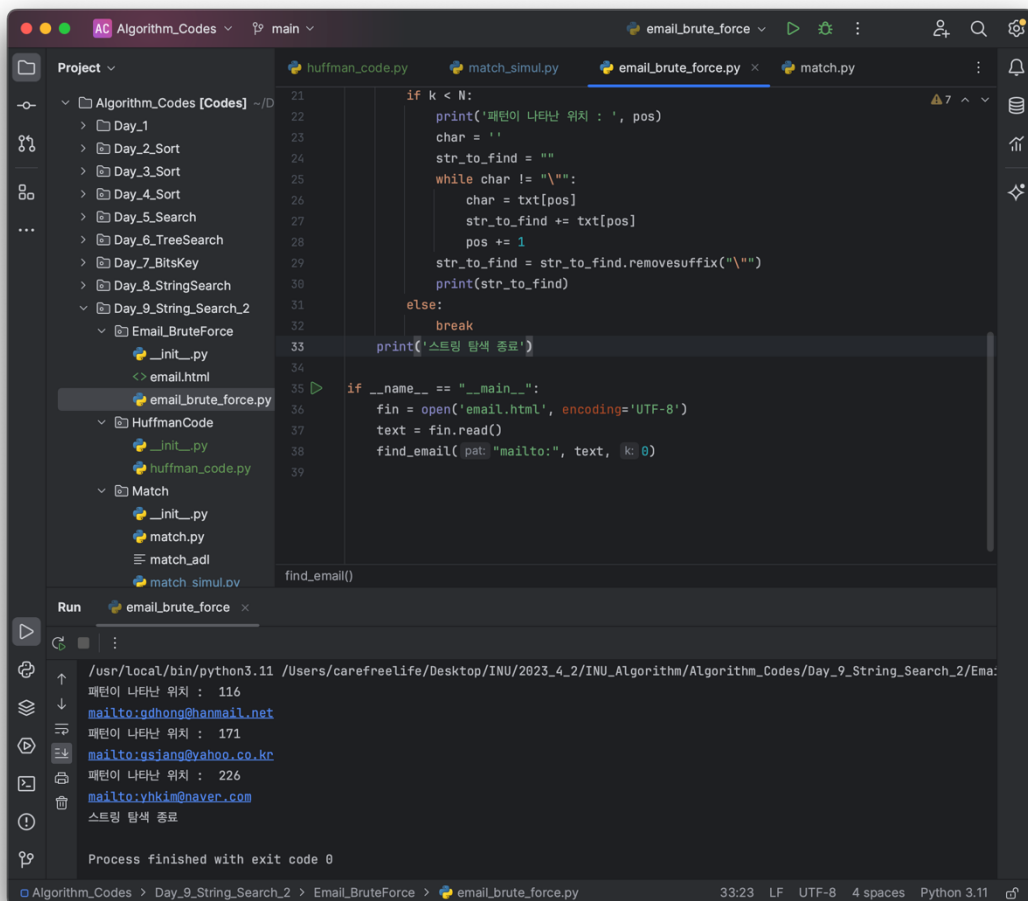
201702797 채승민

1. 다음과 같은 HTML 파일에서 "mailto:" 라는 문자열 뒤에 있는 이메일 주소를 가져와서 한 줄에 하나씩 화면에 출력하는 프로그램을 작성하라.

프로그램은 다음과 같은 순서로 작성한다.

- (1) 먼저 파이썬 소스 파일이 저장된 폴더와 같은 폴더에 email.html 파일을 저장한다.
- (2) email.html 파일의 내용을 변수 text 에 저장한다.
- (3) 직선적 스트링 탐색 알고리즘을 구현한 bruteForce() 함수를 사용하여 "mailto:" 패턴이 나오는 위치를 탐색한다.
- (4) "mailto:" 패턴을 찾으면 "(쌍따옴표)가 나올 때까지 text 에 있는 문자를 모아서 화면에 출력한다.

[실행 결과]



The screenshot shows a Python IDE with a project named 'Algorithm_Codes'. The file explorer on the left shows a directory structure for 'Day_9_String_Search_2' containing 'Email_BruteForce' and 'Match' subdirectories. The main editor displays the code for 'email_brute_force.py'. The code defines a 'find_email()' function that iterates through the content of 'email.html' to find email addresses. The 'Run' output at the bottom shows the execution results, including the file path, the positions of the 'mailto:' pattern, and the extracted email addresses.

```
21 if k < N:
22     print('패턴이 나타난 위치 : ', pos)
23     char = ''
24     str_to_find = ""
25     while char != "\n":
26         char = txt[pos]
27         str_to_find += txt[pos]
28         pos += 1
29     str_to_find = str_to_find.removesuffix("\n")
30     print(str_to_find)
31 else:
32     break
33 print('스트링 탐색 종료')
```

```
35 if __name__ == "__main__":
36     fin = open('email.html', encoding='UTF-8')
37     text = fin.read()
38     find_email(pat: "mailto:", text, k: 0)
39
```

```
find_email()

/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_9_String_Search_2/Ema:
패턴이 나타난 위치 : 116
mailto:gdhong@hanmail.net
패턴이 나타난 위치 : 171
mailto:gsjang@yahoo.co.kr
패턴이 나타난 위치 : 226
mailto:yhkim@naver.com
스트링 탐색 종료

Process finished with exit code 0
```

[소스 코드]

```
def bruteForce(p, t, k):
    M = len(p); N = len(t)
    i, j = k, 0
    while j < M and i < N:
        if t[i] != p[j]:
            i = i - j
            j = -1
        i += 1
        j += 1
    if j == M:
        return i - M
    else:
        return i

def find_email(pat, txt, k):
    M = len(pat); N = len(txt)

    while True:
        pos = bruteForce(pat, txt, k)
        k = pos + M
        if k < N:
            print('패턴이 나타난 위치 : ', pos)
            char = ''
            str_to_find = ""
            while char != "\n":
                char = txt[pos]
                str_to_find += txt[pos]
                pos += 1
            str_to_find = str_to_find.removesuffix("\n")
            print(str_to_find)
        else:
            break
    print('스트링 탐색 종료')

if __name__ == "__main__":
    fin = open('email.html', encoding='UTF-8')
    text = fin.read()
    find_email("mailto:", text, 0)
```

2. 다음 패턴 매칭 알고리즘을 ADL 로 작성하라.

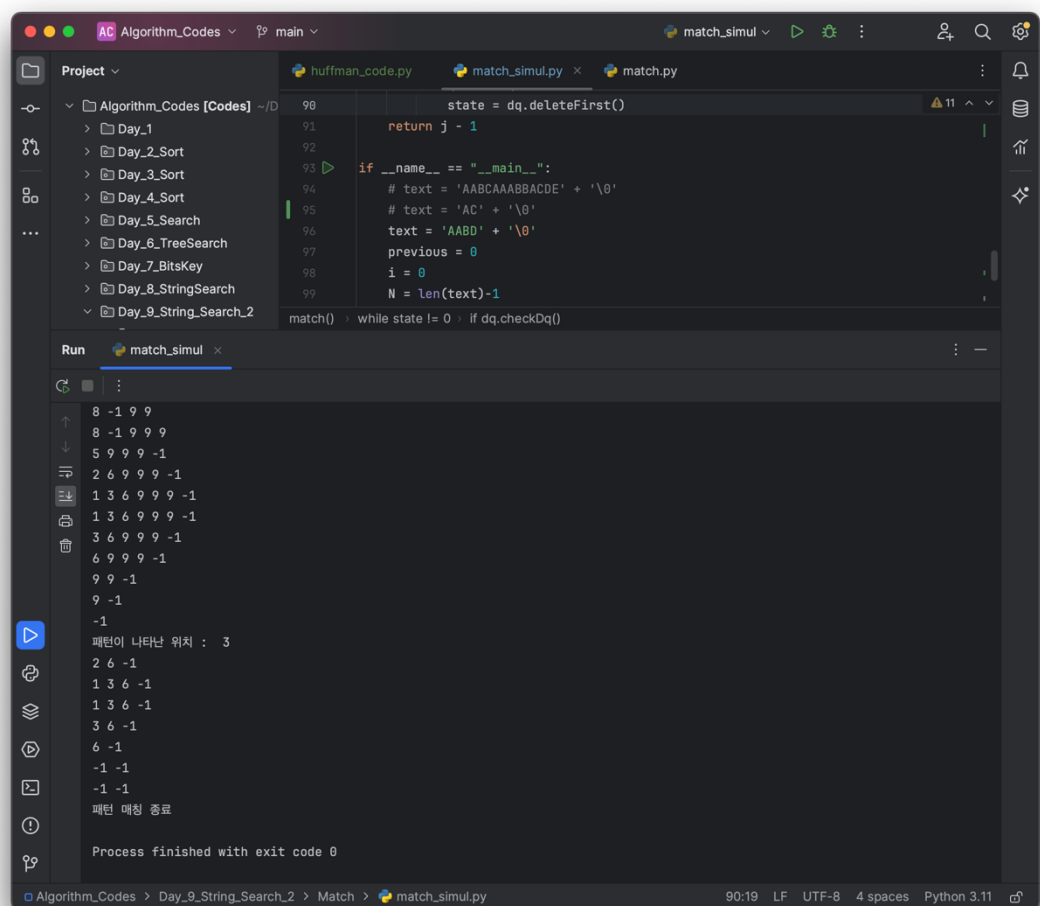
```
match(t[])
dq ← Deque(100);
j ← 0;
N ← t 의 길이 - 1;
state ← next1[0];
dq.insertLast(scan);
while (state ≠ 0) {
  case {
    state = scan:
      j ← j + 1;
      if (dq.isEmpty()) then dq.insertFirst(next1[0]);
      dq.insertLast(scan);
    ch[state] = t[j]:
      dq.insertLast(next1[state]);
    ch[state] = ' ':
      n1 ← next1[state];
      n2 ← next2[state];
      dq.insertFirst(n1);
      if (n1 ≠ n2) then dq.insertFirst(n2);
  }
  if (dq.isEmpty()) then return j;
  if (j > N) then return 0
  state ← dq.deleteFirst()
  if (dq.checkDq()) then state ← dq.deleteFirst();
return j-1;
end match()
```

패턴 매칭 알고리즘을 파이썬으로 작성하고, 다음과 같은 정규식에 대해 정확히 동작하는지 확인해 보라. 패턴을 입력한 다음 패턴을 식별하는 과정에서 데크가 변화하는 과정을 출력하라. 인식되는 패턴 외에도 인식되지 않는 패턴에 대해서도 확인해 보라.

(0) ch, next1, next2 배열의 모습

```
1 scan = -1
2
3 # (A*B+AC)D
4 # ch = [' ', 'A', ' ', ' ', 'B', ' ', ' ', ' ', 'A', 'C', 'D', ' ', ' ']
5 # next1 = [5, 2, 3, 4, 8, 6, 7, 8, 9, 0]
6 # next2 = [5, 2, 1, 4, 8, 2, 7, 8, 9, 0]
7
8 # # (A + B)*C
9 # ch = [' ', ' ', ' ', ' ', ' ', ' ', 'A', 'B', ' ', ' ', 'C', ' ', ' ']
10 # next1 = [1, 2, 3, 4, 6, 6, 2, 8, 0]
11 # next2 = [1, 2, 7, 5, 6, 6, 2, 8, 0]
12
13 # (AB* + A*D)E
14 ch = [' ', ' ', ' ', 'A', ' ', ' ', 'B', 'A', ' ', ' ', 'D', ' ', ' ', 'E', ' ', ' ']
15 next1 = [1, 2, 3, 4, 3, 6, 7, 8, 9, 10, 0]
16 next2 = [1, 6, 3, 8, 3, 6, 7, 8, 9, 10, 0]
```

(1) (A*B + AC)D



The screenshot shows a code editor with a project named 'Algorithm_Codes'. The file 'match_simul.py' is open, showing a function 'match()' and a main block. The output of the program is displayed in the 'Run' console.

```
90 state = dq.deleteFirst()
91 return j - 1
92
93 if __name__ == "__main__":
94     # text = 'AABCAAABBACDE' + '\0'
95     # text = 'AC' + '\0'
96     text = 'AABD' + '\0'
97     previous = 0
98     i = 0
99     N = len(text)-1
100     match()
101     while state != 0:
102         if dq.checkDq():
103             state = dq.deleteFirst()
104             return j - 1
```

Run console output:

```
8 -1 9 9
8 -1 9 9 9
5 9 9 9 -1
2 6 9 9 9 -1
1 3 6 9 9 9 -1
1 3 6 9 9 9 -1
3 6 9 9 9 -1
6 9 9 9 -1
9 9 -1
9 -1
-1
패턴이 나타난 위치 : 3
2 6 -1
1 3 6 -1
1 3 6 -1
3 6 -1
6 -1
-1 -1
-1 -1
패턴 매칭 종료
Process finished with exit code 0
```

(2) $(A + B) * C$

```
Project
├── Algorithm_Codes [Codes] ~/D
│   ├── Day_1
│   ├── Day_2_Sort
│   ├── Day_3_Sort
│   ├── Day_4_Sort
│   ├── Day_5_Search
│   ├── Day_6_TreeSearch
│   ├── Day_7_BitsKey
│   ├── Day_8_StringSearch
│   └── Day_9_String_Search_2
│       ├── Match
│       └── match_simul.py
└── ...

huffman_code.py
match_simul.py
match.py

state = dq.dequeue()
return j - 1

if __name__ == "__main__":
    # text = 'AABCAABBACDE' + '\0'
    text = 'AC' + '\0'
    # text = 'AABD' + '\0'
    previous = 0
    i = 0
    N = len(text)-1
    while True:
        if __name__ == "__main__":

Run
match_simul

5 4 8 8 -1
4 8 8 -1
8 -1
-1
패턴이 나타난 위치 : 1
2 -1
7 3 -1
7 3 -1 8
5 4 -1 8
5 4 -1 8
4 -1 8
1 8 -1
2 8 -1
7 3 8 -1
7 3 8 -1
5 4 8 -1
5 4 8 -1
4 8 -1
-1
패턴 매칭 종료

Process finished with exit code 0

Algorithm_Codes > Day_9_String_Search_2 > Match > match_simul.py
95:23 LF UTF-8 4 spaces Python 3.11
```

(3) (AB* + A*D)E

```
Project
├── Algorithm_Codes [codes]
│   ├── Day_1
│   ├── Day_2_Sort
│   ├── Day_3_Sort
│   ├── Day_4_Sort
│   ├── Day_5_Search
│   ├── Day_6_TreeSearch
│   ├── Day_7_BitsKey
│   ├── Day_8_StringSearch
│   └── Day_9_String_Search_2
│       ├── Email_BruteForce
│       ├── email_brute_force.py
│       ├── email.html
│       ├── HuffmanCode
│       └── init.py
└── match_simul.py
└── match.py
```

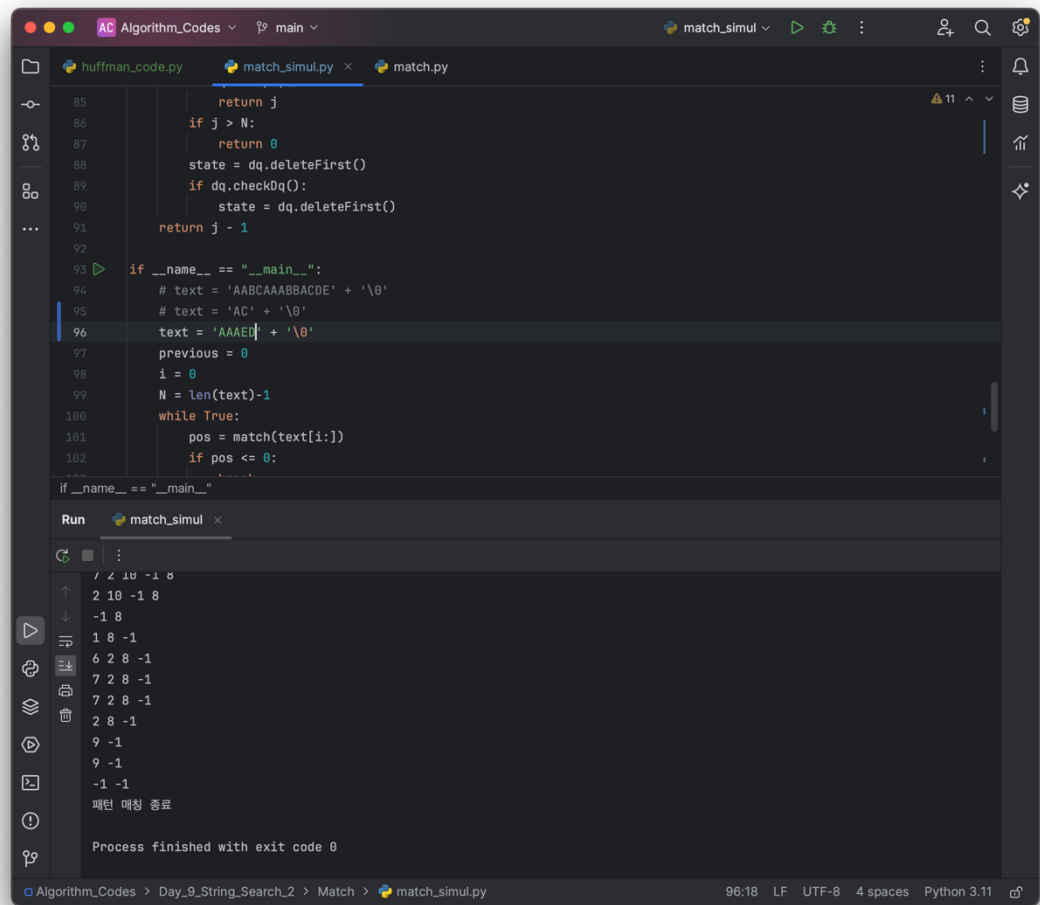
```
88         state = dq.deleteFirst()
89         if dq.checkDq():
90             state = dq.deleteFirst()
91         return j - 1
92
93 if __name__ == "__main__":
94     # text = 'AABCAAABBACDE' + '\0'
95     # text = 'AC' + '\0'
96     text = 'ABE' + '\0'
97     previous = 0
98     i = 0
99     N = len(text)-1
100     while True:
101         pos = match(text[i:])
102         if pos <= 0:
103             break
104
105 if __name__ == "__main__":
```

```
1 10 -1
6 2 10 -1
7 2 10 -1
7 2 10 -1
2 10 -1
-1
패턴이 나타난 위치 : 2
6 2 -1
7 2 -1
7 2 -1
2 -1
-1 -1
-1 -1
패턴 매칭 종료

Process finished with exit code 0
```

Algorithm_Codes > Day_9_String_Search_2 > Match > match_simul.py 96:13 LF UTF-8 4 spaces Python 3.11

(4) 패턴을 찾을 수 없는 경우 예시: (AB* + A*D)E



The screenshot shows a code editor with a Python script and its execution output. The script is named `match_simul.py` and is located in the `Algorithm_Codes` directory. The script defines a function `match` that takes a string `text` and a pattern `pattern` as input. The pattern is `(AB* + A*D)E`. The script then runs the `match` function with the input text `'AAAE'` and the pattern `'(AB* + A*D)E'`. The output of the script is a list of matches, which is empty. The output is displayed in the `Run` panel at the bottom of the editor.

```
85         return j
86     if j > N:
87         return 0
88     state = dq.deleteFirst()
89     if dq.checkDq():
90         state = dq.deleteFirst()
91     return j - 1
92
93 if __name__ == "__main__":
94     # text = 'AABCAABBACDE' + '\0'
95     # text = 'AC' + '\0'
96     text = 'AAAE' + '\0'
97     previous = 0
98     i = 0
99     N = len(text)-1
100     while True:
101         pos = match(text[i:])
102         if pos <= 0:
103             break
104     if __name__ == "__main__":
105         print(pos)
```

Run match_simul x

```
/ 4 10 -1 8
2 10 -1 8
-1 8
1 8 -1
6 2 8 -1
7 2 8 -1
7 2 8 -1
2 8 -1
9 -1
9 -1
-1 -1
패턴 매칭 종료

Process finished with exit code 0
```

- 문자열이 E 로 끝나지 않고 중간에 존재해야할 D 가 가장 마지막에 있는 경우, 패턴을 찾지 못하고 종료하게 된다.

[소스 코드]

```
scan = -1

# (A*B+AC)D
# ch = [' ', 'A', ' ', ' ', 'B', ' ', ' ', 'A', 'C', 'D', ' ']
# next1 = [5, 2, 3, 4, 8, 6, 7, 8, 9, 0]
# next2 = [5, 2, 1, 4, 8, 2, 7, 8, 9, 0]

# # (A + B)*C
# ch = [' ', ' ', ' ', ' ', ' ', ' ', 'A', 'B', ' ', ' ', 'C', ' ']
# next1 = [1, 2, 3, 4, 6, 6, 2, 8, 0]
# next2 = [1, 2, 7, 5, 6, 6, 2, 8, 0]

# (AB* + A*D)E
ch = [' ', ' ', ' ', 'A', ' ', ' ', 'B', 'A', ' ', ' ', 'D', ' ', ' ', 'E', ' ']
next1 = [1, 2, 3, 4, 3, 6, 7, 8, 9, 10, 0]
next2 = [1, 6, 3, 8, 3, 6, 7, 8, 9, 10, 0]

class Deque:
    def __init__(self, size):
        self.deque = []
        self.first = int(size/2)
        self.last = int(size/2)
        for i in range(size):
            self.deque.append(0)

    def insertFirst(self, v):
        self.deque[self.first] = v
        self.first -= 1

    def insertLast(self, v):
        self.last += 1
        self.deque[self.last] = v

    def deleteFirst(self):
        self.deque[self.first] = 0
        self.first += 1
        return self.deque[self.first]

    def isEmpty(self):
        if self.first == self.last:
            return True
        else:
            return False

    def checkDq(self):
        if self.deque[self.first] == 0:
            if self.last - self.first < 2 and self.deque[self.last] ==
scan:
            return False
        elif not self.isEmpty():
            return True
        else:
            return False
    else:
        return False
```

```

def prDq(self, size):
    for i in range(size):
        if self.deque[i] != 0:
            print(self.deque[i], end=' ')
    print()

def match(t):
    dq = Deque(100)
    j = 0
    N = len(t) - 1
    state = next1[0]
    dq.insertLast(scan)
    while state != 0:
        match state:
            case _ if state == scan:
                j += 1
                if dq.isEmpty() == 0:
                    dq.insertFirst(next1[0])
                dq.insertLast(scan)
            case _ if ch[state] == t[j]:
                dq.insertLast(next1[state])
            case _ if ch[state] == ' ':
                n1 = next1[state]
                n2 = next2[state]
                dq.insertFirst(n1)
                if n1 != n2:
                    dq.insertFirst(n2)
    dq.prDq(100)
    if dq.isEmpty():
        return j
    if j > N:
        return 0
    state = dq.deleteFirst()
    if dq.checkDq():
        state = dq.deleteFirst()
    return j - 1

if __name__ == "__main__":
    # text = 'AABCAAABBACDE' + '\0'
    # text = 'AC' + '\0'
    text = 'AAAED' + '\0'
    previous = 0
    i = 0
    N = len(text) - 1
    while True:
        pos = match(text[i:])
        if pos <= 0:
            break
        pos += previous
        i = pos
        if i <= N:
            print('패턴이 나타난 위치 : ', pos)
        else:
            break
        previous = i
    print('패턴 매칭 종료')

```

3. 허프만 트리를 생성해 주는 파이썬 프로그램을 사용하여 문자열 “A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS”에 대한 허프만 코드를 생성하려고 한다.

(1) count[k]와 dad[k]를 출력하여 다음 표와 동일한지 확인해 보라.

k	0	1	2	3	4	5	6	7	9	12	13	14	15	16	18	19	20	21
count[k]	11	3	3	1	2	5	1	2	6	2	4	5	3	1	2	4	3	2
dad[k]	-40	-32	-33	27	-28	36	-27	29	-37	-30	35	37	33	28	31	-35	32	30

k	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
count[k]	2	3	4	4	5	6	6	8	8	10	11	12	16	21	33	37	60
dad[k]	-29	-31	-34	34	-36	-38	38	39	-39	40	41	-41	42	-42	43	-43	0

Count[k], dad[k] 의 출력 모습

```

Run  huffman_code x
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_9_String_Search_2/Huf
Original Text: A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS
count[k]: [11, 3, 3, 1, 2, 5, 1, 2, 0, 6, 0, 0, 2, 4, 5, 3, 1, 0, 2, 4, 3, 2, 0, 0, 0, 0, 0, 2, 3, 4, 4, 5, 6, 6, 8, 8, 10
dad[k]: [-40, -32, -33, 27, -28, 36, -27, 29, 0, -37, 0, 0, -30, 35, 37, 33, 28, 0, 31, -35, 32, 30, 0, 0, 0, 0, 0, 0, -29, -
Encoded Text: 0111111101100110101101101000111001111011011011000100010010111011001001110101110011111000001001100100110111
Decoded Text: A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS
Process finished with exit code 0
  
```

(2) code[k]와 length[k]를 출력하여 다음 표와 동일한지 확인해 보라.

	k	code[k]	length[k]	
	0	7	3	111
A	1	7	4	0111
B	2	5	4	0101
C	3	38	6	100110
D	4	55	6	110111
E	5	12	4	1100
F	6	39	6	100111
G	7	18	5	10010
I	9	1	3	001
L	12	17	5	10001
M	13	10	4	1010
N	14	0	3	000
O	15	4	4	0100
P	16	54	6	110110
R	18	26	5	11010
S	19	11	4	1011
T	20	6	4	0110
U	21	16	5	10000

code[k], length[k] 의 출력 모습

```
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_9_String_Search_2/Huf
Original Text: A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS

count[k]: [11, 3, 3, 1, 2, 5, 1, 2, 0, 6, 0, 0, 2, 4, 5, 3, 1, 0, 2, 4, 3, 2, 0, 0, 0, 0, 0, 2, 3, 4, 4, 5, 6, 6, 8, 8, 10,
dad[k]: [-40, -32, -33, 27, -28, 36, -27, 29, 0, -37, 0, 0, -30, 35, 37, 33, 28, 0, 31, -35, 32, 30, 0, 0, 0, 0, 0, -29, -3

code[k]: [7, 7, 5, 38, 55, 12, 39, 18, 0, 1, 0, 0, 17, 10, 0, 4, 54, 0, 26, 11, 6, 16, 0, 0, 0, 0, 0]
length[k]: [3, 4, 4, 6, 6, 4, 6, 5, 0, 3, 0, 0, 5, 4, 3, 4, 6, 0, 5, 4, 4, 5, 0, 0, 0, 0, 0]

Encoded Text: 01111111011001101010110100011100111101101101000100010010111011001001110101110011110000010011001001101111
Decoded Text: A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS

Process finished with exit code 0
```

(3) findDad(maxIndex, k) 함수를 사용하여 허프만 코드로 인코딩된 스트링을 디코드하는 함수 decode()를 파이썬으로 작성하고, 생성된 허프만 코드가 정확하게 디코드되는지 확인해 보라.

decode() 후 허프만 코드가 정확하게 디코드되는 모습

```
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_9_String_Search_2/Huf
Original Text: A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS

Encoded Text: 011111111011001101101101101000111001111011011011010001000100101110110010011101011100011111000001001100100110111
Decoded Text: A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS

Process finished with exit code 0
```

[소스코드]

```
class PQ:
    def __init__(self):
        self.heap = [0]*100
        self.info = [0]*100
        self.n = 0

    def insert(self, v, x):
        self.n += 1
        i = self.n
        while True:
            if i == 1: break
            if v >= self.heap[int(i/2)]: break
            self.heap[i] = self.heap[int(i/2)]
            self.info[i] = self.info[int(i/2)]
            i = int(i/2)
        self.heap[i] = v
        self.info[i] = x

    def remove(self):
        x = self.info[1]
        temp_v = self.heap[self.n]
        temp_x = self.info[self.n]
        self.n -= 1
        i = 1
        j = 2
        while j <= self.n:
            if (j < self.n) and (self.heap[j] > self.heap[j+1]):
                j += 1
            if temp_v <= self.heap[j]: break
            self.heap[i] = self.heap[j]
            self.info[i] = self.info[j]
            i = j
            j *= 2
        self.heap[i] = temp_v
        self.info[i] = temp_x
        return x
```

```

def isEmpty(self):
    if self.n == 0: return True
    else: return False

def index(c):
    if ord(c) == 32:
        return 0
    else:
        return (ord(c)-64)

def makeHuffman(t, m):
    for i in range(m):
        count[index(t[i])] += 1
    for i in range(27):
        if count[i]:
            pq.insert(count[i], i)
    i = 27
    while not pq.isEmpty():
        t1 = pq.remove()
        t2 = pq.remove()
        dad[i] = 0
        dad[t1] = i
        dad[t2] = -i
        count[i] = count[t1] + count[t2]
        if not pq.isEmpty():
            pq.insert(count[i], i)
        i += 1
    for k in range(27):
        i = x = 0
        j = 1
        if count[k]:
            q = dad[k]
            while q:
                if q < 0:
                    x += j
                    q = -q
                q = dad[q]
                j += j
                i += 1
        code[k] = x
        length[k] = i

def encode(t, m):
    huffman_code = ''
    for j in range(m):
        i = length[index(t[j])]
        while i > 0:
            huffman_code += str((code[index(t[j])]) >> i - 1) & 1)
            i -= 1
    return huffman_code

def char(k):
    if k == 0: return chr(32)
    else: return chr(k+64)

def findDad(max_i, k):
    for i in range(max_i):
        if dad[i - 1] == k:

```

```

        # print(f"dad[i] ({dad[i - 1]}) == k({k})")
        return i
    return -1

# 인코딩된 문자열들을 앞에서부터 읽어들이며 root 노드로부터
# 0 이 나오면 왼쪽 자식으로 이동
# 1 이 나오면 오른쪽 자식으로 이동
# 단말 노드가 나올때까지 이를 반복.
# 단말 노드가 나온 경우 그 노드에 알맞은 알파벳을 출력한 후 다시 루트 노드로
돌아가서 반복
    # k 는 41 부터 시작.
    # 허프만 코드가 0 이면 k 와 같은 절대값을 가진 양수 dad[k] 로 이동.
    # 1 이면 음수 dad[k] 로 이동.
def decode(h):
    decoded_text = ""

    # k 배열 마지막 값 복사.
    k_idx = len(k) - 1
    temp = k[k_idx]
    temp_str = ""
    cp_h = h
    while len(cp_h) != 0:
        # k 와 같은 dad[k] 값이 없을 때 까지 이동.
        # 41
        # 이전 while loop 에서 변경된 k 배열 마지막 값 초기화
        k[k_idx] = temp
        toggle = False
        while k[k_idx] in dad:
            # 띄어쓰기 검출
            if k[k_idx] == 0:
                toggle = not toggle
                break

            # Huffman 코드가 0 인 경우 양수
            if cp_h[0] == '0':
                k[k_idx] = dad.index(k[k_idx])

            # Huffman 코드가 1 인 경우 음수
            else:
                k[k_idx] = dad.index(-(k[k_idx]))
            # Huffman code 첫 문자 삭제
            cp_h = cp_h[1:]

        if toggle:
            decoded_text += ' '
        else:
            decoded_text += chr(k[k_idx] + 64)
    return decoded_text

if __name__ == "__main__":
    # text = 'VISION QUESTION ONION CAPTION GRADUATION EDUCATION'
    text = 'A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF
BITS'
    print(f"\nOriginal Text: {text}\n")

```

```
count = [0]*100
dad = [0]*100
length = [0]*27
code = [0]*27
M = len(text)
pq = PQ()
makeHuffman(text, M)

# print(f"count[k]: {count} \ndad[k]: {dad}\n")

# print(f"code[k]: {code} \nlength[k]: {length}\n")

k = []
for i in range(len(count)):
    if count[i] != 0:
        k.append(i)
h = encode(text, M)
print(f"Encoded Text: {h}\n")
d = decode(h)
print(f"Decoded Text: {d}\n")
```

감사합니다.
201702797 채승민